Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему : «**Створення інструменту для автоматичної адаптації елементів UI до навігації за допомогою ігрових контролерів в iOS**»

Виконав : студент 4-го року навчання,
Спеціальності
122 Комп’ютерні науки
Столяров Владислав Сергійович
Керівник : Франків О.О.,
кандидат фіз.-мат. наук, доцент
Рецензент
Кваліфікаційна робота захищена
з оцінкою
Секретар ЕК
«» 2024 р.

Київ – 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

к.ф.-м.н.

С. С. Гороховський _____

(підпис)

„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

Студенту Столярову В.С. факультету інформатики 4 курсу

Тема: «Створення інструменту для автоматичної адаптації елементів UI до навігації за допомогою ігрових контролерів в iOS»

Зміст текстової частини кваліфікаційної роботи:

Календарний план
Зміст
Анотація
Вступ
1 Інтеграція ігрових контролерів у розробці для iOS
2 Розробка фреймворку GameControllerBinder
3 Публікація та використання фреймворку
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі “____” _____ 2024 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

ТЕМА: «Створення інструменту для автоматичної адаптації елементів UI до навігації за допомогою ігрових контролерів в iOS»

Календарний план виконання роботи:

№	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу	20.10.2023	
2.	Огляд наукової літератури за темою роботи	10.11.2023	
3.	Початок написання фреймворку GameControllerBinder	01.12.2023	
4.	Створення алгоритму для навігації за допомогою ігрових контролерів	27.03.2024	
5.	Початок написання текстової частини кваліфікаційної роботи	20.04.2024	
6.	Завершення та публікація фреймворку в загальний доступ	10.05.2024	
7.	Завершення написання теоретичної частини	19.05.2024	
8.	Оформлення слайдів для доповіді	20.05.2024	
9.	Захист кваліфікаційної роботи	31.05.2024	

Студент Столяров В.С. _____

Керівник Франків О.О. _____

“_____”_____2024 р.

ЗМІСТ

Анотація

В даній роботі розроблено загальнодоступний фреймворк GameControllerBinder на мові програмування Swift, що спрощує інтеграцію геймпадів у додатки iOS та дозволяє керувати інтерфейсом додатків за їх допомогою. Описано роботу фреймворків, зокрема наявний фреймворк для інтеграції геймпадів від Apple - Game Controller. Продемонстровано процес публікації фреймворку на GitHub з подальшою інтеграцією з менеджерами залежностей у Swift. Алгоритм у класі FocusManager ефективно обчислює найближчі сусідні елементи для створення навігаційної мапи екрану. Запропонований протокол Focusable дозволяє керувати різними елементами за допомогою геймпада та змінювати їх зовнішній вигляд при фокусуванні.

Вступ

Стрімкий розвиток мобільних процесорів протягом останніх десятиліть зробив мобільні телефони пристроями, здатними замінити велику кількість інших речей в нашому повсякденному житті, оскільки майже все можна зробити маючи з собою даний мобільний пристрій. Останніми ж роками потужності мобільних процесорів, особливо тих, що випускає компанія Apple стали такими, що телефони від цієї компанії здатні запускати високобюджетні, графічно насичені ігри та складні додатки[1]. Цей технологічний стрибок, а також заохочення компанії Apple зробити свої пристрої ігровими підкреслюється нещодавньою появою на платформах iOS та macOS таких ігор, як «Death Stranding», «Resident Evil 8» та «Resident Evil 4 Remake»[2]. Ці ігри, традиційно зарезервовані для ринків консолей або висококласних ПК, тепер доступні на портативних пристроях, що свідчить про значний зсув у підході ігрової індустрії до мобільних ігор. Ці ігри налаштовані на їхнє керування за допомогою ігрових контролерів, для того, щоб повноцінно керувати ігровим процесом на iPhone потрібно використовувати бездротові ігрові контролери. Проте наразі інтеграція ігрових контролерів у нові або вже існуючі додатки є доволі складним процесом, а попит на це стає все більшим з кожним роком.

Об’єктом дослідження є взаємодія мобільних додатків з зовнішніми апаратними інтерфейсами, зокрема ігровими контролерами.

Предметом дослідження є процес адаптації інтерфейсів мобільних додатків для ефективної взаємодії з ігровими контролерами.

Метою цього дослідження є розробка комплексного фреймворку, який не тільки дозволяє безперешкодно інтегрувати ігрові контролери з iOS-додатками, але й гарантує, що ця інтеграція підтримує інтуїтивно зрозумілу навігацію та доступність. Конкретні завдання, пов'язані з досягненням цієї мети, включають :

- Розуміння існуючої реалізації інтеграції ігрових контролерів до iOS-пристроїв.

- Створення GameControllerBinder для встановлення та керування з'єднаннями між iOS-пристроями та ігровими контролерами.
- Розробка FocusManager для обробки логіки, необхідної для динамічного управління фокусом між елементами інтерфейсу.
- Реалізація протоколу Focusable для стандартизації інтерактивних можливостей різноманітних компонентів інтерфейсу, щоб вони могли ефективно реагувати на вхідні дії контролера.
- Проведення серії експериментів для перевірки ефективності фреймворку та його впливу на користувацький досвід і доступність.

Ця кваліфікаційна робота складається з трьох основних розділів, що слідують за вступом. У першому розділі розглядаються теоретичні основи та сучасні технології, пов’язані з розробкою на мові програмування Swift, використанням фреймворків в iOS та історію розвитку ігрових контролерів. Другий розділ описує процес створення фреймворка GameControllerBinder, включаючи детальний аналіз класів GameControllerBinder та FocusManager, протоколу Focusable та розширень для UIKit елементів. Третій розділ присвячений публікації фреймворка, його інтеграції через SPM та CocoaPods, а також тестуванню продуктивності та корисності у реальних додатках.

Розділ 1. Інтеграція ігрових контролерів у розробці для iOS

1.1 Мова програмування Swift та екосистема розробки на iOS

Мова програмування Swift - це сучасна надійна та інтуїтивно зрозуміла мова програмування, створена компанією Apple у 2014 році для створення додатків для iOS, Mac, Apple TV та Apple Watch. Swift розроблено з акцентом на безпеку, а його синтаксис заохочує розробників писати чистий і читабельний код[3]. Однією з особливостей Swift є підтримка протокольно-орієнтованого програмування (POP), яке сприяє створенню більш гнучких і придатних для повторного використання шаблонів коду порівняно з традиційними об'єктно-орієнтованими моделями програмування.

Ця парадигма в Swift передбачає використання протоколів для визначення набору методів і властивостей, які повинен реалізовувати будь-який відповідний тип. POP стає все більш популярним у розробці додатків для iOS завдяки своїм численним перевагам, включаючи збільшення повторного використання коду, краще розділення завдань, покращення тестування та підвищення продуктивності[4].

```
protocol Identifiable {
    var id: String { get set }
    func identify()
}
```

Рис. 1.1 Створення протоколу у Swift

```
extension Identifiable {
    func identify() {
        print("My ID is \(id).")
    }
}
```

Рис. 1.2 Розширення до протоколу, що визначає типову поведінку функції протоколу

```
struct User: Identifiable {
    var id: String
}

let twostraws = User(id: "twostraws")
twostraws.identify()
```

Рис. 1.3 Створення структури, що відповідає протоколу. Ця структура автоматично отримує типову поведінку функції

Протоколи у Swift особливо потужні у поєднанні з широкими можливостями середовища розробки для iOS. Ця екосистема надає розробникам повний набір інструментів та API, призначених для використання повного потенціалу сучасного обладнання. Використовуючи протокольно-орієнтовану природу Swift, розробники можуть більш ефективно інтегрувати такі функції, як обробка дотиків, жести та зовнішні аксесуари, такі як ігрові контролери - ключові компоненти у створенні доступних та інтерактивних додатків.

Середовище розробки для iOS доступне лише для macOS. Воно включає в себе надає повний набір бібліотек, фреймворків, інтерфейсів прикладного програмування (API) та інших ресурсів, які розробники можуть використовувати для створення додатків для пристроїв на iOS та iPadOS. Вони також можуть використовувати SDK для створення додатків для комп'ютерів Mac з технологією Apple Silicon. Оскільки програми працюють у власному форматі iOS, їх не потрібно модифікувати для macOS. Сама ж розробка додатків виконується в середовищі Xcode.

1.2 Фреймворки в iOS: Розширення функціональності додатків

Фреймворками в розробці на мові програмування Swift називають ієрархічну директорію, яка містить спільні ресурси, такі як динамічна спільна бібліотека, nib-файли, файли зображень, локалізовані рядки, заголовні файли та довідкову документацію в одному пакеті. Кілька програм можуть використовувати всі ці ресурси одночасно. Система завантажує їх у пам'ять у міру необхідності і, коли це можливо,

ділить одну копію ресурсу між усіма програмами[5]. Це невід'ємна частина розробки додатків на будь-яку платформу Apple, адже навіть для написання базового коду на Swift необхідно імпортувати такий фреймворк як Foundation, що надає базовий рівень функціональності для додатків і фреймворків, включаючи зберігання і збереження даних, обробку тексту, обчислення дати і часу, сортування і фільтрацію, а також роботу з мережею. Класи, протоколи та типи даних, визначені Foundation, використовуються у всіх SDK для macOS, iOS, watchOS та tvOS.

Існують фреймворки написані самою компанією Apple так і фреймворки, написані сторонніми розробниками. Фреймворки Apple - це динамічні бібліотеки, які попередньо встановлені в системі iOS, що означає, що вони легко доступні без збільшення розміру файлу програми. Для розробників така інтеграція означає підвищення продуктивності додатків та пришвидшення часу завантаження, оскільки необхідний код вже існує на пристрої та є спільним для всіх додатків, які його використовують. Такий підхід не лише оптимізує використання пам'яті, але й забезпечує дотримання суворих стандартів продуктивності Apple.

Імпортуючи фреймворк Apple у Swift-проект, розробники просто використовують оператор «import» на початку Swift-файлів. Наприклад,
«import UIKit» часто зустрічається у верхній частині Swift-файлів, що стосуються компонентів користувацького інтерфейсу. Цей єдиний рядок дозволяє розробникам отримати доступ до тисяч заздалегідь написаних класів і методів, розроблених спеціально для створення надійних і ефективних iOS-додатків.

І навпаки, сторонні фреймворки, які не входять до системи iOS, вимагають від розробників пройти процес налаштування за допомогою менеджерів залежностей, таких як CocoaPods, Carthage або Swift Package Manager. Ці інструменти допомагають керувати додаванням, оновленням та видаленням сторонніх бібліотек. Хоча це і додає додатковий рівень складності до процесу розробки, але дозволяє підвищити гнучкість та включити інноваційні функції, які недоступні в екосистемі Apple.

Однією з важливих переваг сторонніх фреймворків є прозорість та залучення спільноти, яку вони пропонують. На відміну від фреймворків Apple із закритим кодом, багато сторонніх бібліотек мають відкритий вихідний код, що забезпечує видимість

коду, який запускає функціональні можливості. Ця відкритість заохочує колективний підхід до розробки, коли розробники можуть долучатися до вдосконалення бібліотеки, виправляти помилки та розширювати можливості. Однак це також може створювати ризики, пов'язані з якістю, безпекою та підтримкою коду, оскільки відповідальність за оновлення та захист коду часто лежить на спільноті розробників, які можуть не мати таких ресурсів, як Apple.

Фреймворки є основою для розширення можливостей додатків, розроблених в екосистемі iOS. Власні фреймворки Apple призначені для максимізації продуктивності та інтеграції додатків з апаратними та програмними функціями iOS. Вони надають розробникам доступ до складних функцій, таких як обробка мультимедіа, безпечне керування даними та розширені можливості користувацького інтерфейсу. Наприклад, такі фреймворки, як AVFoundation та MapKit, дозволяють розробникам легко інтегрувати мультимедійні дані та функції геолокації у свої додатки, використовуючи весь потенціал апаратного забезпечення Apple.

З іншого боку, сторонні фреймворки доповнюють пропозиції Apple, вирішуючи завдання, які можуть потребувати більш спеціалізованих або спрощених підходів до реалізації. Ці фреймворки часто зосереджені на підвищенні продуктивності розробників та зменшенні складності кодових баз. Наприклад, Alamofire значно спрощує управління мережевим зв'язком, абстрагуючись від значної частини шаблонного коду, пов'язаного з HTTP-запитами. Аналогічно, фреймворки на кшталт KeychainSwift надають більш доступний інтерфейс для безпечного зберігання даних, полегшуючи використання складних механізмів безпеки Apple.

Більше того, поява сторонніх фреймворків реактивного програмування, таких як RxSwift, запроваджує альтернативні парадигми програмування, дозволяючи розробникам більш ефективно керувати асинхронними подіями та потоками даних. Це особливо корисно для складних додатків, де управління змінами стану та взаємодією з користувачами може стати громіздким за допомогою традиційних підходів. Хоча Apple представила власний фреймворк Combine для підтримки реактивного програмування, сторонні рішення продовжують процвітати, пропонуючи додаткові рівні абстракції та можливості інтеграції.

По суті, в той час як фреймворки Apple відкривають прямий доступ до можливостей пристрою і забезпечують оптимальну продуктивність, сторонні фреймворки розширюють набір інструментів, доступних для розробників, сприяючи більш універсальному і часто більш простому процесу розробки. Ця стратегія збагачує платформу розробки iOS, пропонуючи поєднання оптимізації продуктивності та гнучкості розробки, що має вирішальне значення для створення сучасних, багатофункціональних додатків.

1.3 Ігрові контролери як інструменти для керування користувацьким інтерфейсом

Ігрові контролери з’явились з появою перших ігрових консолей на початку 1960-х років. Це були пристрої, метою яких було керування ігровим процесом. З кожним новим поколінням консолей геймпади набували все більш сучасного вигляду, коли до них додавались додаткові кнопки, аналогові стіки та спускові гачки або ж тригери[6].

Рис. 1.4 Оригінальний контролер NES (1983 рік)

Починаючи з третього покоління, ігрові консолі здобувають всесвітню популярність, головно ж консоллю того покоління була Nintendo Entertainment System,

яка продалась тиражом більше ніж 60 млн копій. Вже в цьому поколінні ігрових консолей контролери були частково схожі на те, що ми маємо сьогодні. Наприклад ігровий контролер NES вже мав D-pad (скорочено від Directional Pad) – кнопку, що має вигляд хрестовини, яка об’єднує в собі чотири напрямки і призначена для керування напрямом руху, а також дві круглі кнопки з позначками «A» та «B». Окрім цього контролер містив допоміжні кнопки «Select» та «Start». Такі ж самі кнопки тільки під іншими назвами або в інших місцях можна і побачити на сучасних ігрових контролерах приставок дев’ятого покоління.

Рис. 1.5 Ігровий контролер DualSense для Playstation 5 (2020 рік)

Сучасні ігрові контролери вже містять в собі набагато більше елементів для керування інтерфейсом ігрових консолей та ігровим процесом в відеоіграх. За приклад можна взяти ігровий контролер розроблений компанією Sony для Playstation 5, але який можна використовувати і у поєднанні з мобільними пристроями, і також з комп’ютером. В ньому також як і в контролері NES є D-pad, замість двох круглих кнопок – чотири, які позначені геометричними фігурами (хрест, коло, трикутник, квадрат), дві допоміжні кнопки. Але окрім цього з еволюцією ігрових контролерів в ньому ще з’явились два аналагових стіка, дві кнопки нагорі геймпада що також називаються бамперами, два тригери, що знаходяться за двома бамперами, сенсорною панеллю посередині і кнопкою PlayStation знизу. В інших сучасних ігрових

контролерах присутні майже всі елементи що й на DualSense окрім сенсорної панелі та кнопки Playstation.

І якщо багато додаткових кнопок з’явилось тільки щоб покращити взаємодію гравця у відеоіграх, оскільки ігровий процес ставав все більш і більш комплексним з часом, то основні кнопки та D-pad, що були присутні на контролерах понад 40 років, дуже гарно допомагають керувати користувацьким інтерфейсом і поза іграми. І оскільки наразі контролери підтримують функцію бездротового з’єднання через Bluetooth – то них можна підключити до будь-якого мобільного пристрою або персонального комп’ютеру та керувати процесом в різних додатках використовуючи кнопки контролерів.

1.4 Фреймворк від Apple: Game Controller

До багатьох пристроїв від Apple можна підключити майже всі сучасні ігрові контролери, і якщо в іграх є підтримка контролерів, то ігровим процесом можна керувати використовуючи один з будь-яких сучасних бездротових ігрових контролерів.

Для того, щоб розробникам додати можливість підтримки фізичних ігрових контролерів Apple створила фреймворк під назвою Game Controller[7]. Він надає великий набір інструментів для інтеграції підтримки ігрових контролерів у їхні додатки для iOS, macOS, tvOS та visionOS. Ігрові контролери включають продукти сторонніх виробників, такі як DualShock 4, DualSense і Xbox, а також миша, клавіатура, Siri Remote і гоночні керма.

Основні можливості даного фреймворку включають в себе :

1. Виявлення та підключення контролерів :

Фреймворк автоматично виявляє та підключає сумісні ігрові контролери. Розробники можуть використовувати сповіщення для обробки підключень та відключень контролерів, забезпечуючи динамічну реакцію програми на зміни доступних пристроїв введення.

2. Обробка входів контролера:

Фреймворк Game Controller забезпечує стандартизований спосіб обробки вводу з різних частин контролера, зокрема кнопок, D-pad, паличок, тригерів і сенсорних панелей. Ця стандартизація дозволяє розробникам писати послідовний код, який працює на різних типах контролерів. Розробники можуть отримати доступ до стану входів контролера і реагувати на такі події, як натискання кнопок і рухи джойстика.

3. Профілі та персоналізація :

Фреймворк підтримує різні профілі контролерів, наприклад, розширений профіль геймпада, який включає додаткові кнопки та елементи керування які присутні на окремих типах ігрових контролерів, як-от DualSense від PlayStation 5 або XboxController від консолей Xbox, порівняно зі стандартним профілем геймпада. Це дозволяє розробникам адаптувати обробку вводу у своєму додатку до конкретних можливостей підключеного контролера.

Також фреймворк підтримує тактильний зворотній зв’язок (haptic feedback) – функцію доступну на деяких контролерах Xbox та PlayStation, а також адаптивні тригери, які доступні тільки на DualSense.

Звичайно, як і будь-який інший фреймворк від Apple – цей може розширити функціональність додатку, проте імплементування даних функцій доволі громіздкий процес, який вимагає від розробників написання багатьох строк коду – щодо підключення контролеру, отримання повідомлень від входів контролера, реалізацій функцій при натиску якоїсь кнопки контролеру, а оскільки контролери від PlayStation та Xbox мають свої окремі функції та додаткові кнопки, то розробникам треба робити додаткові налаштування щоб мати доступ саме до цих додаткових кнопок чи функцій, що мають ці контролери.

Окрім цього, даний фреймворк не дає жодних допоміжних функцій для навігації по різних UI елементах, отже для того, щоб керувати додатком за допомогою ігрового контролеру розробникам доводиться писати додатковий код, задля того, щоб це працювало правильно та інтуїтивно зрозуміло – такі додаткові труднощі, які виникають під час додавання інтеграції ігрових контролерів у вже додатки, стають ще однією причиною того, що кількість додатків, особливо не ігрових, що підтримують ці пристрої введення – дуже мала.

1.5 Висновки

Підсумовуючи вищезазначене, стає очевидним те, що підключення ігрових контролерів до пристроїв iOS залишається складним завданням. Наразі, окрім власного фреймворку Apple, немає готових рішень, які дозволяють розробникам легко додавати підтримку ігрових контролерів. Інтеграція цього фреймворку вимагає значного обсягу коду, особливо при реалізації навігації та управління додатком за допомогою контролера.

Фреймворки в Swift служать не лише для розширення функціональності додатків через використання можливостей пристроїв, але й для полегшення процесу розробки застосунків. Сторонні фреймворки зазвичай зосереджені на підвищенні продуктивності розробників та зменшенні складності написаного коду. Використовуючи ці принципи, можна створити інструмент, який значно спростить підключення контролерів до пристроїв iOS та зробить інтеграцію та навігацію доступною для будь-якого додатка, навіть вже існуючого.

Для цього, можна використовувати можливості мови програмування Swift, зокрема протоколи, які є ключовим елементом у Swift. Протоколи дозволяють створювати гнучкі та масштабовані рішення, які легко інтегруються з існуючими API та фреймворками. Створення власного фреймворку, який забезпечить легке підключення ігрових контролерів та їх інтеграцію в додатки, дозволить розробникам уникнути написання великого обсягу коду та частіше інтегрувати ігрові контролери в розроблені додатки.

Розділ 2. Розробка фреймворку GameControllerBinder

2.1 Загальна інформація про фреймворк

Фреймворк GameControllerBinder призначений для спрощення інтеграції ігрових контролерів у додатки для iOS, полегшуючи розробникам додавання навігації по власному додатку та прив’язки входів контролера до дій в додатку. Цей фреймворк мінімізує складнощі, пов'язані з обробкою входів ігрових контролерів, і забезпечує структурований підхід до управління фокусом і взаємодією в додатку. Він цілком і повністю написаний на основі фреймворку від Apple – Game Controller.

Основними компонентами фреймворку є :

1. Клас GameControllerBinder :

Головний клас фреймворку. Даний клас керує підключенням ігрових контролерів, обробляє події вводу та прив'язує ці події до дій або елементів інтерфейсу користувача у додатку.

2. Клас FocusManager :

Цей клас відповідає за керування системою фокусування у програмі. Він відстежує елементи, які можуть бути у фокусі, обчислює їхніх найближчих сусідів та обробляє переходи фокусу на основі даних, що вводяться користувачем з ігрового контролера.

3. Протокол Focusable :

Протокол, якому повинні відповідати елементи інтерфейсу, щоб бути розпізнаними та керованими класом FocusManager. Цей протокол визначає властивості та методи, необхідні для того, щоб елемент вважався сфокусованим, такі як name, isFocusable, globalFrame, focus(), unfocus() та simulateTap().

4. Розширення до елементів UIKit :

Розширення надаються для різних елементів UIKit, таких як UIControl, UITableView, UISearchBar і UITextView, що дозволяє їм відповідати протоколу Focusable і безперешкодно взаємодіяти з FocusManager.

Ці всі основні компоненти надають таку функціональність цьому фреймворку:

- Автоматична реєстрація елементів, що можуть бути сфокусованими: фреймворк автоматично виявляє та реєструє всі елементи, що фокусуються, у вікні, дозволяючи розробникам швидко налаштовувати навігацію за допомогою ігрових контролерів з мінімальним кодом.

- Обробка входів контролера для навігації по елементах, які є Focusable: даний фреймворк також надає функцію що дозволяє додати керування додатком за допомогою D-pad, який присутній на будь-якому контролері.

- Прив’язка входів контролера до дій в самому додатку: розробники можуть прив’язати будь-який елемент будь-якого можливого контролера до певних дій або елементів інтерфейсу. Фреймворк дозволяє повноцінно і легко з кнопками не тільки стандартних контролерів, а й з кнопками контролерів Xbox та PlayStation, а також з тригерами та аналоговими стіками, повністю контролюючи їхнє положення.

З цією функціональністю даний фреймворк робить інтеграцію ігрових контролерів більш доступною та спрощеною для iOS-розробників. У наступних підрозділах буде більш детально описано деталі реалізації основних компонентів даного фреймворку та його інтеграцію в додатки.

2.2 Клас GameControllerBinder

Клас GameControllerBinder є основним у цьому фреймворку. Він керує прив'язкою входів ігрового контролера до певних дій та елементів інтерфейсу в iOS-додатку. Цей клас полегшує інтеграцію різних ігрових контролерів, таких як контролери PlayStation та Xbox, надаючи розробникам простий у використанні інтерфейс для роботи з входами ігрових контролерів. Також він містить в собі екземпляр класу FocusManager, який керує всіма елементами, які можуть бути у фокусі, а для розробників він надає функції, завдяки яким можна легко додати навігацію за допомогою ігрового контролеру до свого додатку.

GameControllerBinder містить в собі також декілька важливих загальнодоступних перерахувань (enum), які полегшують розробникам взаємодію з усіма можливими елементами контролерів усіх типів, що підтримує Game Controller.

Ці перерахування включать в себе:

- ButtonName: Визначає різні назви кнопок для ігрових контролерів (наприклад: buttonA, buttonB).
- TriggerName: Визначає назви тригерів для ігрових контролерів.
- ThumbstickName: Визначає назви аналагових стіків для ігрових контролерів.
- ControllerType: Вказує тип ігрового контролера (наприклад: dualSense, dualShock).
- PlayStationButtonName: Визначає конкретні назви кнопок, які є тільки у контролерів PlayStation.
- XboxButtonName: Визначає специфічні назви кнопок, які є тільки у контролерів Xbox.

Використовуючи саме такі перерахування під час розробки додатку не буде виникати проблем щоб взаємодіяти саме з потрібним елементом контролеру без необхідності взаємодії з класом GCController та його різними профілями як extendedGamepad, GCDualShockGamepad, GCDualSenseGamepad та GCXboxGamepad щоб мати можливість отримати доступ до необхідної кнопки яка присутня саме на DualSense чи DualShock.

Окрім цього GameControllerBinder містить в собі приватні атрибути, які визначають все необхідне для того, щоб загальнодоступні методи цього класу працювали коректно. Основні атрибути цього класу, це :

- buttonActionBindings: Словник для зберігання дій, пов'язаних з натисканням кнопки.
- buttonActionReleaseBindings: Словник для зберігання дій, пов’язаних з натисканням та відпусканням кнопки.
- triggerActionBindings: Словник для зберігання дій, пов’язаних зі зміною стану тригерів.
- thumbstickActionBindings: Словник для зберігання дій, пов’язаних зі зміною для аналогових стіків.
- focusManager: Екземпляр класу FocusManager для керування фокусом елементів інтерфейсу.

- desiredInitialFocusElementName: Зберігає назву елементу, який повинен бути у фокусі за потреби розробника.

Ці словники створено для того, щоб потім легко зчитувати входи, що приходять на якусь конкретну кнопку чи інший елемент з контролера, і потім прив’язувати конкретну дію, вже встановлену розробником через загальнодоступну функцію до конкретної кнопки. Також окрім зазначених словників, є ще окремо словники для кнопок, які присутні лише на контролерах Xbox та PlayStation. Екземпляр класу FocusManager потрібен, оскільки вже через нього буде проходити процес запису та керування усіма елементами, які можуть бути у фокусі, і оскільки сам клас internal, тобто розробник не має до нього доступу, він повинен бути присутнім і ініціалізованим для кожного екземпляру класу GameControllerBinder, який буде створеним розробником в межах ViewController. Останній же атрибут, який зберігає назву елементу, який повинен бути у фокусі потрібен для того, щоб в ситуації, коли жоден контролер не підключений до пристрою, при майбутньому підключенні цього пристрою, була можливість сфокусувати саме необхідний розробником елемент. З останнього речення також стає зрозумілим те, що цей фреймворк керує станом того, чи є в конкретний елемент якийсь підключений контролер, і в залежності від цього стану або показує якийсь елемент таким, що він є у фокусі або ні.

Разом із цим необхідно зазначити, що трапляється при ініціалізації екземпляру класу GameControllerBinder, а саме те що під час її налаштовуються спостерігачі сповіщень для виявлення під'єднання та від'єднання ігрових контролерів.

```
/// Initializes a new GameControllerBinder instance.
/// It sets up notification observers for when game controllers connect or disconnect.
public init() {
    NotificationCenter.default.addObserver(self, selector:
        #selector(controllerConnected), name: .GCControllerDidConnect, object: nil)
    NotificationCenter.default.addObserver(self, selector:
        #selector(controllerDisconnected), name: .GCControllerDidDisconnect, object: nil)
}
```

Рис. 2.1 Ініціалізація GameControllerBinder

Після того, як екземпляр класу ініціалізован, цей поточний екземпляр стає спостерігачем для сповіщень щодо того, коли геймпад під’єднано до пристрою, та коли від’єднано від пристрою, і як тільки одне з цих двох сповіщень приходить, викликається метод або controllerConnected, або controllerDisconnected, і саме під час виклику цих методів йде налаштування усіх атрибутів цього класу.

Найголовніше для розробника, це саме ті методи та атрибути класу, до яких в нього буде доступ під час роботи з даних фреймворком. В даному фреймворку таких методів та атрибутів – 15.

Атрибути, до яких має доступ розробник, це:

- isConnected – атрибут типу Boolean, що вказує на те, чи підключений на даний момент контролер.
- controllerType – атрибут типу ControllerType, що вказує на тип ігрового контролеру, якщо такий є підключеним, інакше має значення nil.

Що стосується загальнодоступних методів класу до прив’язки дій до входів ігрового контролера, то основні методи це:

- bindButtonToAction(): Прив'язує замикання (closure) до певного натискання кнопки або до певного натискання та відпускання кнопки.
- bindButtonToUIElement(): Прив'язує кнопку контролера до конкретної UIButton та її дії
- bindTriggerToAction(): Прив'язує замикання до певного тригера.
- bindThumbstickToAction(): Прив'язує замикання до певного аналогового стіку.

Під час їхнього використання розробник обирає будь-який елемент із усіх можливих через перерахування та в замиканні вказує те, що повинно відбуватись під час цього входу контролера. Робота з тригерами та аналоговими стіками відрізняється від кнопок, адже якщо в кнопках два стани – натиснута чи ні, то в тригера є ступінь його натискання, і в замиканні можна працювати з цим значенням. Так само з аналоговими стіками, оскільки їх можна рухати по колу, то в замиканні можна працювати з двома значеннями – наскільки від центрального положення цей стік відхилений по горизонталі, та наскільки – по вертикалі.

Для інтеграції системи керування фокусом у класі GameControllerBinder також є загальнодоступні методи для реєстрації елементів інтерфейсу, що фокусуються та налаштування того, за допомогою чого буде виконуватись навігація по цим focusable елементам:

- registerAllFocusableSubviews(): Реєструє всі можливі елементи, які можна сфокусувати у даному екрані.
- setInitialFocus(): Встановлює початковий фокус на певному елементі інтерфейсу.
- setupDefaultDirectionalInputHandlers(): Налаштовує елементи контролера для навігації по елементам інтерфейсу (D-pad та аналогові стіки).
- setupTapActionInputHandler(): Налаштовує певну кнопку контролера під час натиску якої буде симулюватись дія натискання на елемент, який наразі у фокусі

Усі ці методи розробник має написати у методі viewDidAppear() у конкретному ViewController задля того, що все коректно співпрацювало та можна було б спокійно використовувати ігровий контролер для переміщення та взаємодії між елементами інтерфейсу.

Останнє, але не менш важливе це допоміжні функції, які викликаються під час роботи методу controllerConnected() задля налаштування усіх необхідних входів контролера, виявлення типу контролера та встановлення початкового фокусу на певному елементі.

Це все і становить клас GameControllerBinder, екземпляр якого розробник створює у своєму ViewController та методи якого використовує для більш простого налаштування певних входів контролера або ж автоматичного налаштування щодо керування контролером усіх можливих елементів інтерфейсу.

2.3 Протокол Focusable та розширення елементів UIKit

Під час розробки фреймворку GameControllerBinder стало очевидно, що потрібен стандартизований спосіб керування фокусом для різних елементів

інтерфейсу. Оскільки Swift – як це було зазначено раніше, це протокольно-орієнтована мова програмування, два ці фактори призвели до створення протоколу Focusable. Основною метою цього протоколу є створення спільного підходу до елементів інтерфейсу, які можуть отримувати фокус, що забезпечує безперешкодну навігацію та взаємодію за допомогою ігрових контролерів.

Отже створення такого протоколу задовольнило такі потреби :

1. Стандартизація:

Визначаючи набір властивостей і методів, які повинні реалізовувати всі елементи, що фокусуються, протокол Focusable стандартизує управління фокусом у різних компонентах інтерфейсу. Ця стандартизація спрощує процес розробки та забезпечує узгоджену поведінку.

2. Гнучкість:

Протокольно-орієнтована специфіка Swift полегшує розширення існуючих класів та структур. Протокол Focusable використовує цю можливість, дозволяючи будь-якому підкласу UIView відповідати протоколу та бути таким, що підлягає можливості керування за допомогою контролеру.

3. Навігація:

У контексті потреб, які вирішує даний фреймворк навігація по елементах інтерфейсу має вирішальне значення. Протокол Focusable надає необхідні методи та властивості для керування зміною фокусу, що робить можливим навігацію по інтерфейсу додатку за допомогою ігрових контролерів.

```swift
public protocol Focusable {
    var name: String { get set }
    var isFocusable: Bool { get }
    var globalFrame: CGRect { get }
    func focus()
    func unfocus()
    func simulateTap()
}
```

Рис. 2.2 Визначення протоколу Focusable

У типів, які мають відповідати цьому протоколу повинні бути такі атрибути та методи :

- name: Унікальний ідентифікатор елементу, який використовується з метою відстеження елемента у алгоритмах
- isFocusable: Визначає, чи може елемент наразі отримати фокус.
- globalFrame: Рамка елемента в координатному просторі головного вікна програми.
- focus(): Метод, що викликається, коли елемент отримує фокус, щоб налаштувати його зовнішній вигляд.
- unfocus(): Метод, що викликається, коли елемент втрачає фокус, щоб скасувати будь-які зміни, зроблені в focus().
- simulateTap(): Метод, що імітує жест дотику до елемента.

Оскільки всі елементи, які створюються та додаються на головне вікно додатку під час розробки додатку на UIKit є підкласами UIView[8], то під час реєстрації усіх можливих елементів з головного вікна збираються усі елементи, що належать класу UIView і з них вже фільтруються лише ті елементи, чиї типи відповідають протоколу Focusable. І для того, щоб деякі з існуючих підкласів UIView відповідали даному протоколу, було створено розширення для таких підкласів, використовуючи функціонал Swift. Ці розширення реалізують необхідні властивості та методи, що дозволяє цим елементам безперешкодно брати участь у системі керування фокусом.

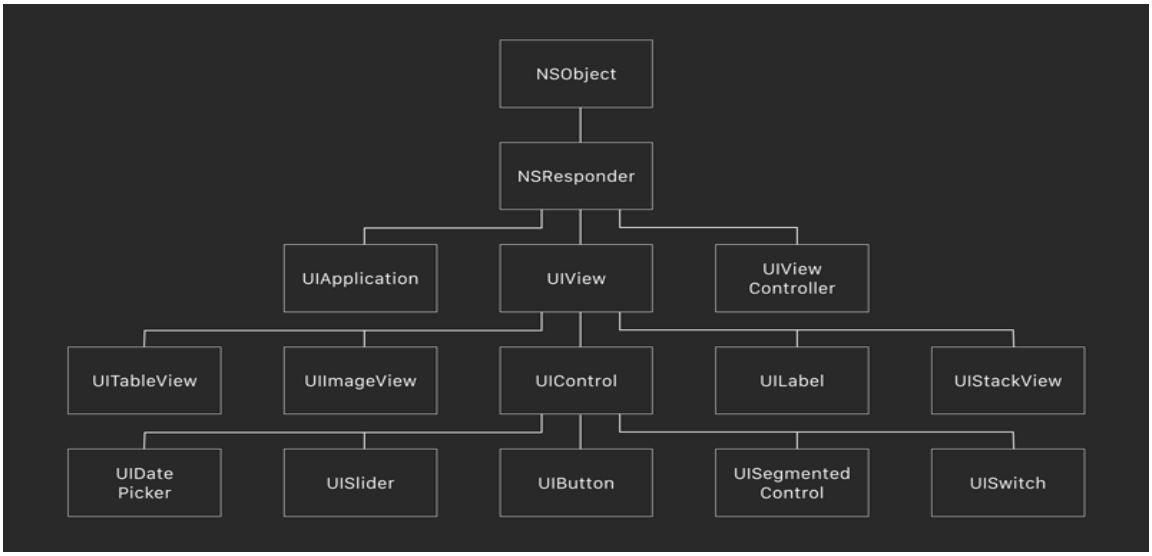

Рис. 2.3 Ієрархія класів UIKit

Звичайно, далеко не всі підкласи UIView мають відповідати протоколу Focusable, оскільки не усіма цими підкласами можна і треба керувати. Основною логікою додавання розширень був факт того, чи можна на ці елементи натискати в програмах і чи буде відбуватися щось після натиску. Саме через це наприклад, UILabel не отримав розширення, оскільки це такий елемент, що відображає один або декілька рядків інформаційного тексту, тобто при натиску на нього нічого не відбувається і сенсу такому елементу бути керованим за допомогою контролера немає. Тому саме такі підкласи UIView мають розширення : UITableView, UISearchBar, UITextView а також UIControl, тобто і всі підкласи UIControl такі як : UIButton, UISwitch, UIStepper, UITextField та інші. Ці всі підкласи є інтерактивними, тобто такими з якими можна взаємодіяти, але оскільки цей протокол загальнодоступний з даним фреймворком, то розробник може додати розширення до будь-якого підкласу UIView, зробивши його Focusable.

Окрім цього було створено окремо клас EdgeElement. Це спеціалізована реалізація протоколу Focusable, призначена для обробки крайніх випадків, коли елементи інтерфейсу повинні взаємодіяти з системою управління фокусуванням, але не підходять для типових випадків використання, охоплених іншими елементами. Цей окремий клас буде необхідним, коли треба буде будувати навігаційну мапу екрану, для розуміння того, де який елемент знаходиться. В будь-яких інших випадках цей клас ніяк не використовуватиметься.

```
final class EdgeElement: Focusable {
    var name: String = "Edge"
    var isFocusable: Bool = false
    var globalFrame: CGRect = .zero // Edge elements don't have a frame.

    func focus() { /* Do nothing */ }
    func unfocus() { /* Do nothing */ }

    public func simulateTap() {/* Do nothing */ }
}
```

Рис. 2.4 Реалізація класу EdgeElement

2.4 Клас FocusManager

Даний клас є важливою складовою даного фреймворку GameControllerBinder. Його головна мета – контролювати елементи, які можуть бути у фокусі, в межах користувацького інтерфейсу додатку, додаючи також навігацію та інтеракцію з даними об’єктами. Саме в цьому класі розроблено алгоритм будування навігаційної мапи екрану.

Перед тим, як перейти до самої структури класу та його атрибутів та методів, важливо розуміти те, як працює керування інтерфейсом за допомогою ігрового контролера. По-перше, завжди повинен бути елемент, який є у фокусі, щоб користувач розумів з яким елементом він наразі взаємодіє. По-друге, зміна елементу, який є у фокусі може бути здійснена у чотирьох напрямках – ліворуч, праворуч, вгору та вниз, оскільки кнопка D-pad на геймпаді має саме чотири напрямки і вона зазвичай використовується для керування інтерфейсом.

Отже, для побудування навігаційної мапи для кожного елементу, який може бути у фокусі треба розуміти який інший елемент серед існуючих є найближчим до цього в усіх чотирьох напрямках. Для цього було створено клас FocusableElement.

```
private final class FocusableElement {

    var thisElement : Focusable
    var nearestRight : Focusable? = nil
    var nearestLeft : Focusable? = nil
    var nearestBottom : Focusable? = nil
    var nearestTop : Focusable? = nil

    init(element : Focusable) {
        self.thisElement = element
    }

}
```

Рис. 2.5 Реалізація класу FocusableElement

Цей клас зберігає інформацію не тільки про конкретний елемент, а й ще про елементи, які є найближчими до цього елементу у всіх можливих чотирьох напрямках. Під час ініціалізації цього класу, екземпляр класу має інформацію лише про цей елемент, інші ж атрибути класу мають значення nil. Після ж того як буде побудована навігаційна мапа екрану у всіх атрибутів будуть якісь значення, але не nil. Саме для цього було створено окремо клас EdgeElement. Якщо елемент на екрані є крайнім або з якогось боку, то для цього елементу найближчим елементом з того боку буде EdgeElement – оскільки він також є Focusable. Це зроблено для, того щоб далі працювати з навігаційною мапою було легше і було чітке розуміння того, що у якогось елементу немає найближчих елементів по якійсь бік. Окрім цього було створено перерахування FocusDirection, для того, щоб було легше працювати з напрямками при знаходженні найближчих елементів.

В самому ж класі FocusManager присутні такі атрибути як :

- focusableElements: Масив екземплярів FocusableElement.
- currentFocusedElement: Атрибут, який відстежує поточний сфокусований елемент.
- currentFocusedIndexPaths: Словник, який відстежує поточний сфокусований рядок у екземплярах UITableView, забезпечуючи точну навігацію у представленнях таблиць.

При додаванні елементів типу Focusable у масив елементів типу FocusableElement перевіряється, чи даний елемент isFocusable, і якщо так, то вже він ініціалізує екземпляр FocusableElement. Після цього постає задача побудування навігаційної мапи серед усіх доступних елементів. Для цього перш за все треба однозначно визначити як по відношенню один до іншого знаходяться елементи.

По-перше, у розробці додатків саме на пристрої iOS початком координат є лівий верхній кут екрану[9]. Чим нижче елемент знаходиться, тим значення його y-координати більше, чим правіше знаходиться елемент, тим значення його x-координати більше. В кожного елементу на екрані є рамка, що представляє собою прямокутник зі сторонами, паралельними осям координат. Для розуміння положення

елементу на екрані у нього є атрибут globalFrame. У цього атрибуту є такі обчислені геометричні атрибути як:

- minX – Найменше значення x-координати прямокутника. Тобто це значення x-координати для кожної точки лівої сторони прямокутника.
- midX – Значення x-координати центру прямокутника.
- maxX – Найбільше значення x-координати прямокутника. Тобто це значення x-координати для кожної точки правої сторони прямокутника.

Такі ж самі атрибути є у globalFrame, що стосуються найменшого, найбільшого та значення центру y-координати в прямокутника, де значення найменшої y-координати – це верхня сторона прямокутника, а значення найбільшої y-координати – це нижня сторона прямокутника

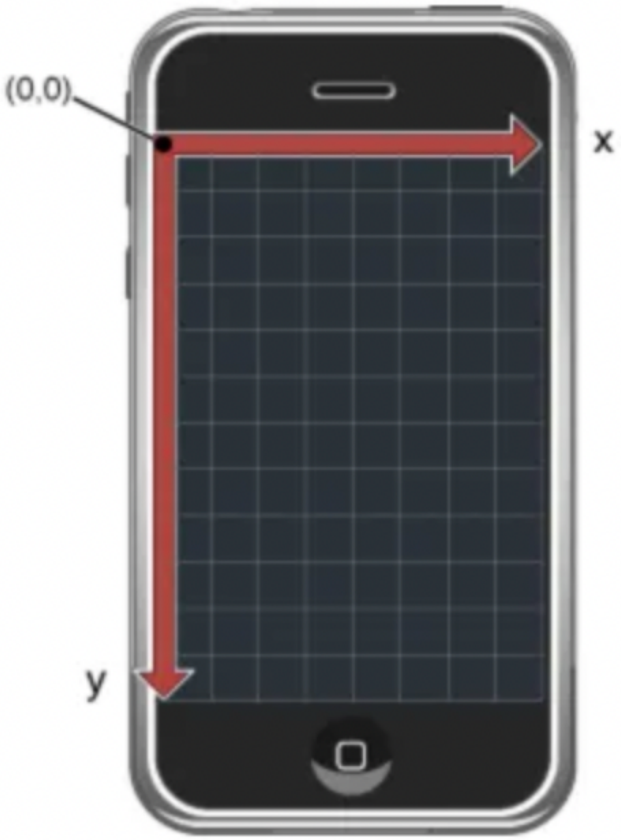

Рис. 2.6 Система координат в UIKit

По-друге, щоб запобігти двозначності, щодо розташування одного елементу відносно іншого було розроблено такі правила :

1. Елемент А знаходиться знизу елемента Б, якщо значення атрибуту minY елемента А більше або дорівнює значенню атрибуту maxY елемента Б.
2. Інакше, елемент А знаходиться зверху елемента Б, якщо значення атрибуту maxY елемента А менше або дорівнює значенню атрибуту minY елемента Б.
3. Інакше, елемент А знаходиться праворуч елемента Б, якщо значення атрибуту minX елемента А більше або дорівнює значенню атрибуту maxX елемента Б.
4. В усіх інших залишившихся випадках елемент А знаходиться ліворуч елемента Б.

Ці правила однозначно визначають положення елементів відносно один іншого. Також варто зазначити, що за умови розположення елементів діагонально один відносно іншого, тобто коли елемент одночасно і зверху і праворуч, наприклад, в цьому випадку береться значення зверху, тобто в даних правилах вертикальне розположення елементів є більш пріорітетним за горизонтальне розположення елементів. Відповідно до цих правил було розроблено метод класу під назвою position(), який як параметри приймає два елементи типу Focusable, а повертає значення типу FocusDirection.

Окрім визначення положення елементів, звичайно треба ще перевіряти чи є наприклад елемент В ближчим до елементу А ніж елемент Б в зазначеному напрямку. Для цього також було створено метод класу під назвою checkIfNearest(), який як параметри приймає три елементи типу Focusable, та один елемент типу FocusDirection, а повертає значення типу Boolean. В самому ж методі для кожного з чотирьох значень FocusDirection перевіряється, чи є елемент ближчим чи ні. Розглянемо перевірку для значення FocusDirection right, тобто чи є елемент В ближчим до елементу А, ніж елемент Б праворуч :

1. Якщо значення атрибуту minX елемента В менше ніж значення minX елемента Б, то повертається true.
2. Інакше, якщо начення атрибуту minX елемента В більше ніж значення minX елемента Б, то повертається false.

3. Інакше, якщо значення атрибутів minX для цих двох елементів однакове, то рахуються відстані від точки правого верхнього кута елемента А до точок середин лівих сторін елементів Б та В, і за умови що відстань до середини сторони елемента В менша або дорівнює відстані до середини сторони елемента Б, повертається true, інакше повертається false.

Таким самим чином проходить перевірка для будь-якого із зазначених напрямків. Також для більш ефективної роботи методу для знаходження найближчих елементів для кожної з сторін набагато простіше працювати з впорядкованим масивом значень FocusableElement, тому було також розроблено методи сортування елементів горизонтально та вертикально.

При сортуванні елементів горизонтально спочатку враховується значення атрибуту minX, тобто чим лівіше елемент, тим раніше він знаходиться у масиві, якщо значення атрибуту однакове для двох елементів, враховується значення maxY, тобто чим вище знаходиться елемент тим раніше він буде знаходитись в масиві. При сортуванні ж елементів вертикально спочатку враховується значення minY, що вищі елементи знаходились раніше у масиву, а при рівності значень цього атрибуту враховується значення maxX, тобто чим лівіше знаходиться елемент, тим раніше він знаходиться у масиві.

Головним же методом у класі FocusManager, який використовуючі усі вище описані допоміжні методи будує навігаційну мапу екрану є метод calculateNearestNeighbors(), мета якого заповнити значення усі атрибути класу FocusableElement, а саме nearestLeft, nearestRight, nearestTop та nearestBottom. Для цього перш за все перевіряється кількість елементів масиву, і якщо їх менше трьох, то вручну заповняються значення атрибутів класу для елементів масиву, інакше йде заповнення цих атрибутів згідно з розробленим алгоритмом, мета якого за найменшу кількість циклів присвоїти правильні значення усім атрибутам.

Спочатку, масив сортується вертикально. Після такого сортування постає мета надати правильні значення атрибутам nearestTop та nearestBottom для усіх елементів масиву. Алгоритм присвоєння значень виглядає таким чином:

1. Одразу після сортування для першого елементу для атрибуту nearestTop надається значення EdgeElement, а для останнього елементу для атрибуту nearestBottom також надається EdgeElement, оскільки ці елементи є крайніми з двох боків, і для цих елементів не може бути елементів що знаходяться вище, якщо це стосується першого елементу, та нижче, якщо це стосується останнього елементу.
2. Оскільки згідно з встановлених правил, перевага надається вертикальному положенню елементів аніж горизонтальному, завжди, коли елемент знаходиться вище за інший метод, що визначає розташування цього елементу буде повертати значення up. Тому спочатку перед основним циклом йде допоміжний який розпочинається з передостаннього елементу, і продовжується допоки елемент з якимось індексом у масиві не розташований зверху над останнім елементом згідно з методу position(). При цьому, якщо елемент розположений не зверху, для його атрибуту nearestBottom надається значення EdgeElement, оскільки якщо такі елементи знаходяться не зверху останнього елементу, значить для них також немає елементів знизу.
3. Основний же цикл триває з першого елементу допоки для елементу з певним індексом значення атрибуту nearestBottom дорівнює nil. Тобто в найгіршому випадку цикл триватиме до передостаннього елементу, інакше ж він може перерватись раніше. Під час цього основного циклу й надаються усі значення для атрибутів. Але перед тим як знаходити найближчі елементи знизу та зверху, якщо поточний елемент є першим, то допоки елемент з якимось індексом у масиві не знаходиться знизу по відношенню до цього елементу, для таких елементів для атрибуту nearestTop надається значення EdgeElement з таких самих причин, які були зазначені у пункті 2. Після цього вже йде основний пошук.
4. Спочатку за допомогою циклу while шукається елемент який знаходиться знизу відносно поточного елементу. Коли такий елемент знайдений то якщо метод checkIfNearest() для цього елементу відносно поточного елементу і значення його атрибуту nearestBottom для позиції down повертає true, то атрибуту

nearestBottom для поточного елементу надається значення знайденого елементу. В цей же час під час схожої перевірки за допомогою методу checkIfNearest() за умови валідної перевірки для атрибуту nearestTop для знайденого елементу надається значення поточного елементу.

5. Після того, як елемент був знайдений, та правильні значення були присвоєні, всередені основного циклу також є внутрішній цикл. Він спрацьовує лише тоді коли індекс знайденого елементу не дорівнює індексу останнього елементу масиву. Цей внутрішній цикл працює з наступного елементу по відношенню до знайденого допоки елемент з конкретним індексом не знаходиться знизу по відношенню до знайденого елементу, тобто мета цього циклу виявити елементи, які знаходяться на одній горизонтальній лінії. Якщо такі елементи знайдено то найголовніше це для них після перевірки checkIfNearest() для атрибутів nearestTop надається значення поточного елементу, і для поточного елементу за умови валідної перевірки для атрибуту nearestBottom надається значення одного з цих елементів.
6. Після внутрішнього циклу, який може і не виконуватись змінюється значення індекса поточного елементу на 1 і робота основного циклу розпочинається зпочатку.

Після такого алгоритму для усіх елементів масиву будуть надані значення nearestTop та nearestBottom. Після цього масив сортується горизонтально для надання значень атрибутам nearestLeft та nearestRight. Алгоритм же присвоєння значень буде трохи відрізнятись від того, яким він був після вертикального сортування :

1. Перший крок тут нічим не відрізняється від того, що було зроблено після вертикального сортування з єдиною зміною що замість значень nearestBottom надається значення для nearestRight, а замість nearestTop – nearestLeft.
2. Оскільки, як було зазначено раніше, перевага надається вертикальному положенню елементів, то не завжди, якщо елемент знаходиться правіше за інший, метод position() повертатиме значення right. Тому в цьому випадку не можна йти таким самим шляхом як і до цього, тому тут одразу йде основний цикл який розпочинається з першого елементу та завершується останнім.

3. Під час цього циклу шукається елемент, який знаходиться по відношенню до поточного праворуч, і так само, якщо такий елемент було знайдено, то після перевірки для його атрибуту nearestLeft надається значення поточного елементу. Так само для значення атрибуту nearestRight для поточного елементу після валідної перевірки надається значення знайденого елементу.
4. В основному циклі так само є внутрішній цикл мета якого така ж сама що і була у такого ж циклу після вертикального сортування – виявити елементи які знаходяться на одній вертикальній лінії. Проте окрім цього, ще одна умова спрацювання циклу, це те, що метод position() повинен повернути значення right для знайдених елементів, оскільки не завжди якщо елемент знаходиться праворуч, його позиція відносно поточного буде такою самою. Але якщо ці умови спрацьовують, то таким же чином після валідних перевірок надаються значення поточного елементу для атрибуту nearestleft для знайдених елементів, і значення одного з елементів для атрибуту nearestRight поточного елементу.
5. Після роботи внутрішнього циклу, якщо для поточного елементу так і не було знайдено елементу праворуч від нього, то для атрибуту nearestRight надається значення EdgeElement і цикл розпочинає свою роботу знову вже з наступного елемента.
6. Після завершення роботи основного циклу викликається ще один цикл по всіх елементах масиву, який перевіряє значення атрибутів nearestLeft елементів масиву. За умови, якщо значення атрибутів дорівнює nil, їм надаються значення EdgeElement.

Таким чином після роботи цих двох алгоритмів після двох сортувань усі атрибути елементів масиву отримують значення, а значить навігаційна мапа екрану побудована, отже тепер можно її використовути для того, щоб переміщуватись між різними елементами.

Отже після того, як було зареєстровано усі елементи, які можуть бути у фокусі викликається метод calculateNearestNeighbors() після виклику якого вже можна ставити фокус на будь-який з доступних елементів, або якщо фокус не встановлено вручну, фокус буде встановлено автоматично на найвищий елемент серед усіх. Після

цього вже можна керувати поточним елементом у фокусі. Для такого керування у класі FocusManager існують такі методи:

- clearFocus() : Знімає фокус з поточно сфокусованого елемента і надає атрибуту currentFocusedElement значення nil.
- changeFocus(): Змінює фокус на наступний елемент на основі вказаного напрямку (вгору, вниз, ліворуч, праворуч). Обробляє переходи між елементами та керує входом і виходом фокусу з комірок UITableView.
- updateFocus(): Оновлює візуальний фокус до нового елемента, який можна сфокусувати. Знімає фокус з поточного елемента і встановлює фокус на новому елементі.
- handleEnteringTableView(): Керує переходом фокусу при вході в UITableView, визначаючи відповідну точку входу на основі напрямку та оновлюючи фокус. Наприклад якщо перехід в UITableView трапляється згори, праворуч або ліворуч, то фокусується перша комірка UITableView, а якщо перехід знизу, то фокусується остання комірка.
- updateFocusToTableView(): Візуально оновлює фокус до певної комірки в UITableView, гарантуючи, що сфокусована комірка буде виділена і UITableView буде прокручено, якщо ця комірка було невидима до цього.
- removeFocusFromTableView(): Видаляє візуальний фокус з певної комірки в UITableView.
- handleLeavingTableView(): Керує переходом фокусу при виході з певної комірки UITableView або за межі UITableView або на іншу комірку цієї UITableView, забезпечуючи належне керування фокусом, коли користувач виходить з представлення таблиці.
- simulateTapOnFocusedElement(): Імітує подію натискання на поточно сфокусованому елементі. Якщо сфокусованим елементом є UITableView, викликає метод didSelectRowAt для сфокусованої комірки. Для інших елементів, що фокусуються, викликає метод simulateTap для виконання відповідної дії.

Усі ці методи і формують клас FocusManager, забезпечуючи правильне та інтуїтивно-зрозуміле керування інтерфейсом за допомогою ігрового контролеру. Цей клас разом із класом GameControllerBinder, протоколом Focusable і створює весь фреймворк GameControllerBinder.

2.5 Можливості для вдосконалення

Поточна реалізація фреймворку GameControllerBinder хоч і надає розробникам значну функціональність, особливо в можливостях легкого керування усіма входами контролера і їх прив’язкою до дій в самому додатку, проте в деяких аспектах даний фреймворк може бути більш вдосконалим у майбутніх версіях.

Насамперед це стосується розширення підримки даного фреймворку для додатків, які розроблені за допомогою SwiftUI. Цей фреймворк є одним з двох, наряду з UIKit, за допомогою яких можна розробляти додатки на iOS і наразі GameControllerBinder розрахований лише для інтеграції в додатки які побудовані з використанням UIKit. Для цього потрібно адаптувати існуючу логіку та протоколи керування фокусом, щоб вони безперешкодно працювали з компонентами SwiftUI.

Окрім цього, в майбутній версії даного фреймворку можна додати можливість кастомізувати елементи, які знаходяться у фокусі. Дозвіл розробникам налаштовувати зовнішній вигляд елементів у фокусі, наприклад, змінювати колір, розмір або додавати анімацію, забезпечить більшу гнучкість в адаптації фреймворку до різних дизайнів і тем інтерфейсу користувача. Також можна буде додати більше розширень до підкласів UIView, наприклад до UICollectionView, щоб більше елементів з самого фреймворку були пристосовані до керування за допомогою контролера і ще можливість адаптуватись під динамічні зміни елементів на екрані, коли їхні позиції змінюються або вони зникають взагалі.

Підсумовуючи, треба зазначити, що з усіма можливими нововведеними зазначеними вище, даний фреймворк стане універсальним інструментом для інтеграції ігрових контролерів у додатки на iOS, хоча й поточна версія фреймворку

вже надає потужний набір можливостей для інтеграції контролерів, та керування додатком за їх допомогою.

Розділ 3. Публікація та використання фреймворку

3.1 Публікація фреймворку та інтеграція з менеджерами залежностей

Для того, щоб фреймворк став загальнодоступним та можливим для використання іншими розробниками, необхідно зробити декілька речей[10]:

- Написати документацію до фреймворку для кожного із загальнодоступних класів, методів та протоколів.
- Доробити цей фреймворк, щоб зробити його завантажуваним через такі менеджери залежностей як CocoaPods та Swift Package Manager.
- Опублікувати його на GitHub створивши ще Readme.md файл та ліцензію.

Документація є необхідною складовою загальнодоступного фреймворку, оскільки розробники, що будуть його використовувати, повинні це робити ефективно та розуміти для чого кожна функція існує і де вона має викликатись.

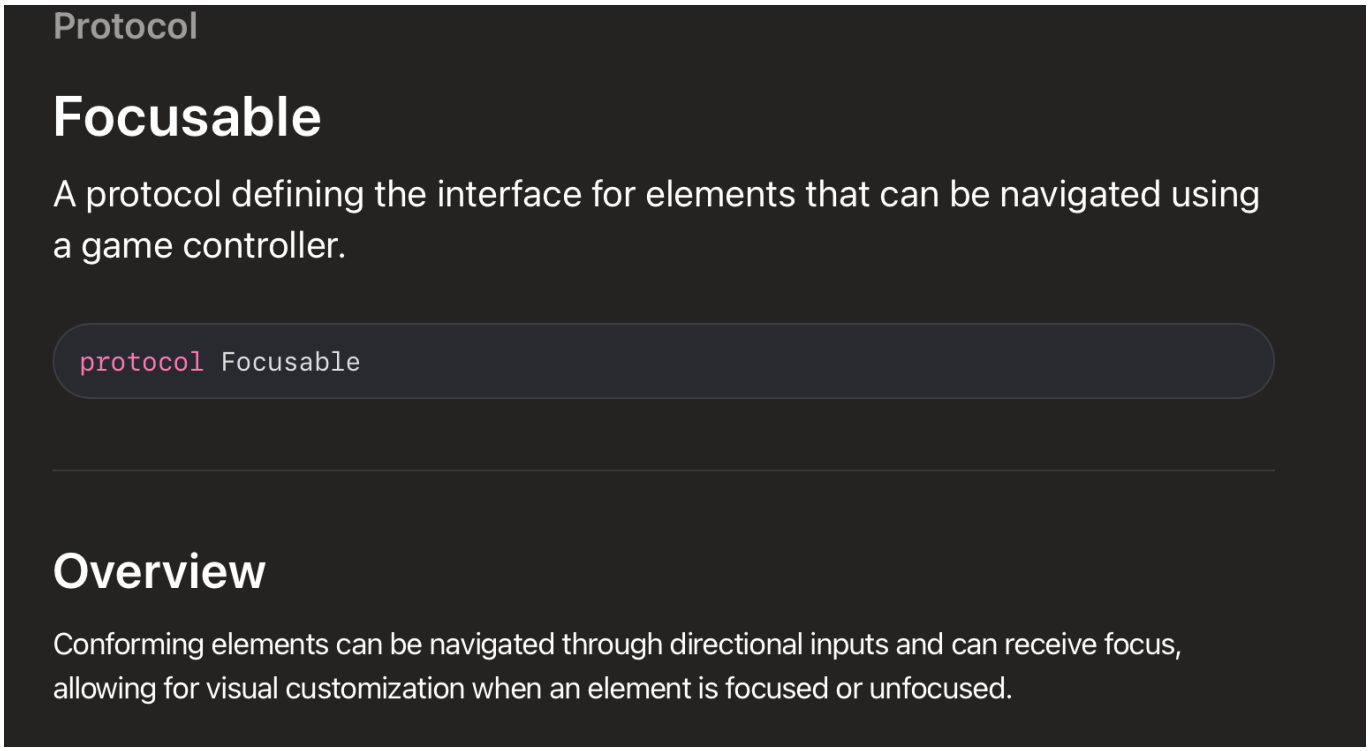

Рис. 3.1 Задокументований протокол Focusable в Apple`s Developer Documentation

Таким чином було задокументовано клас GameControllerBinder, усі перерахування, які належать цьому класу, а також усі загальнодоступні методи та атрибути класу. Окрім цього було задокументовано протокол Focusable і всі його методи та атрибути.

Щоб гарантувати, що фреймворк GameControllerBinder можна буде легко інтегрувати в інші iOS-проекти, було обрано два популярні методи розповсюдження: CocoaPods та Swift Package Manager (SPM). Ці методи широко використовуються у спільноті розробників iOS і надають розробникам безперешкодний спосіб включення зовнішніх бібліотек у свої проекти.

Для того, щоб фреймворк був доступний через CocoaPods, треба створити файл podspec. Цей файл містить метадані про фреймворк, такі як назва, версія, резюме, опис, ліцензія, автори, сумісність з платформами та вихідні файли. Після створення та перевірки файлу podspec фреймворк можна опублікувати в репозиторії CocoaPods за допомогою команди pod trunk push. Це робить фреймворк доступним для встановлення через CocoaPods.

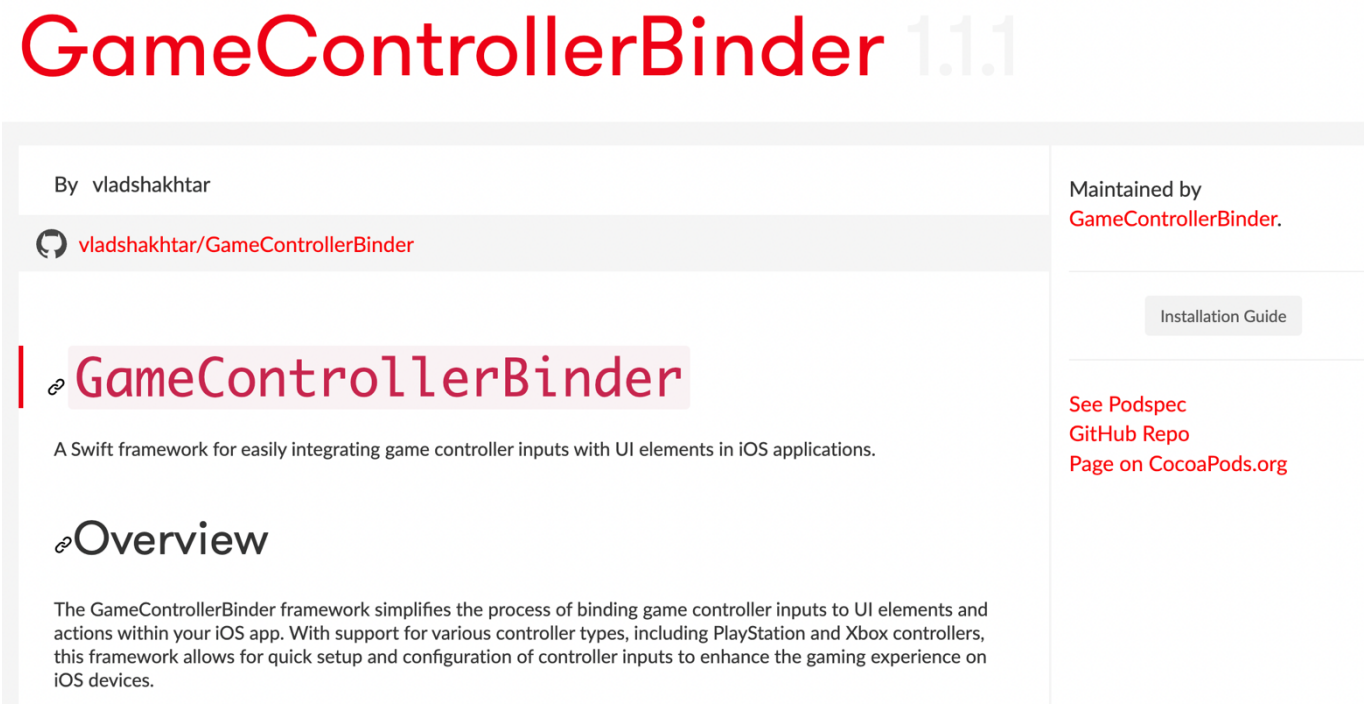

Рис. 3.2 Сторінка фреймворку на cocoapods.org

Для роботи з SPM перш за все треба створити файл Package.swift. Подібно до файлу podspec, файл Package.swift містить метадані про фреймворк, включаючи його назву, платформи, продукти та об’єкти. Щоб зробити фреймворк доступним через SPM, репозиторій GitHub,в якому розташований даний фреймворк потрібно позначити номером версії (наприклад, v1.1.0). Цей тег дозволяє розробникам вказати версію фреймворку для включення у свої проекти.

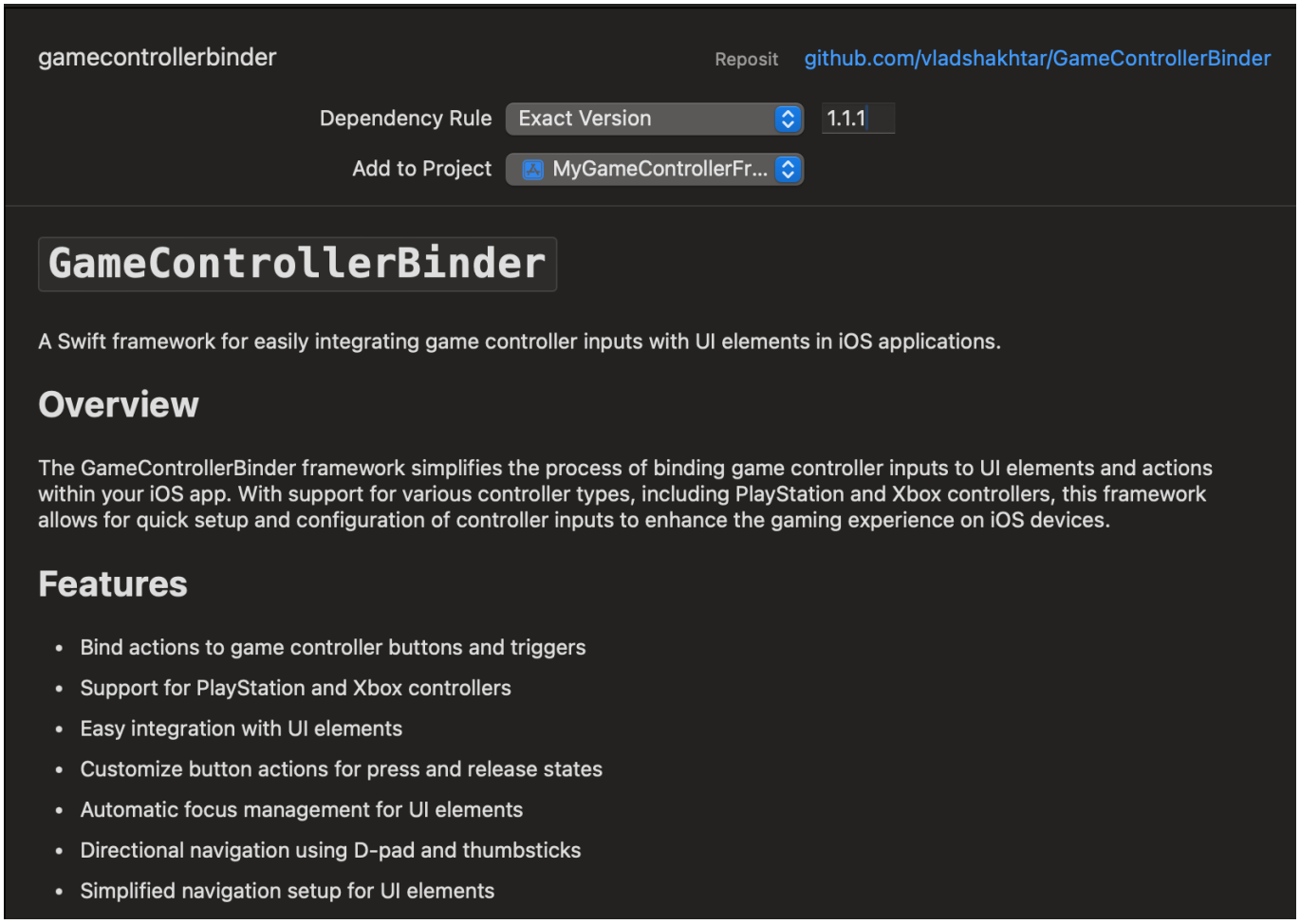

Рис. 3.3 Сторінка фреймворку під час його додавання через SPM

Також весь код фреймворку доступний на репозиторії в GitHub за посиланням: https://github.com/vladshakhtar/GameControllerBinder. Окрім цього було створено файл README.md. В цьому файлі надано загальну інформацію про фреймворк, його можливості, інструкція з інтеграції цього фреймворку за допомогою SPM та CocoaPods, приклади фрагментів коду з використанням фреймворку, та інформація про ліцензію фреймворку. Оскільки цей фреймворк загальнодоступний, то він

надається за ліцензією MIT. Файл ліцензії, створений на основі стандартного шаблону, включений до репозиторію фреймворку.

3.2 Використання фреймворку в тестовому додатку

Для демонстрування роботи фреймворку та ефективності його алгоритмів було створено невеличкий тестовий додаток, в якому буде продемонстровано швидкість роботи алгоритму по будуванню навігаційної мапи екрану, а також як виглядають елементи інтерфейсу які знаходяться у фокусі.

Сам додаток містить десять кнопок (UIButton), а також UITableView з сьома комірками, дві з котрих залишаються невидимими через розмір клітини. Також цей додаток використовуватиме модифікований фреймворк, в якому після будування навігаційної мапи буде виведена в консоль інформація про всі елементи та їх сусідів, для того, щоб продемонстувати правильну роботу алгоритму.

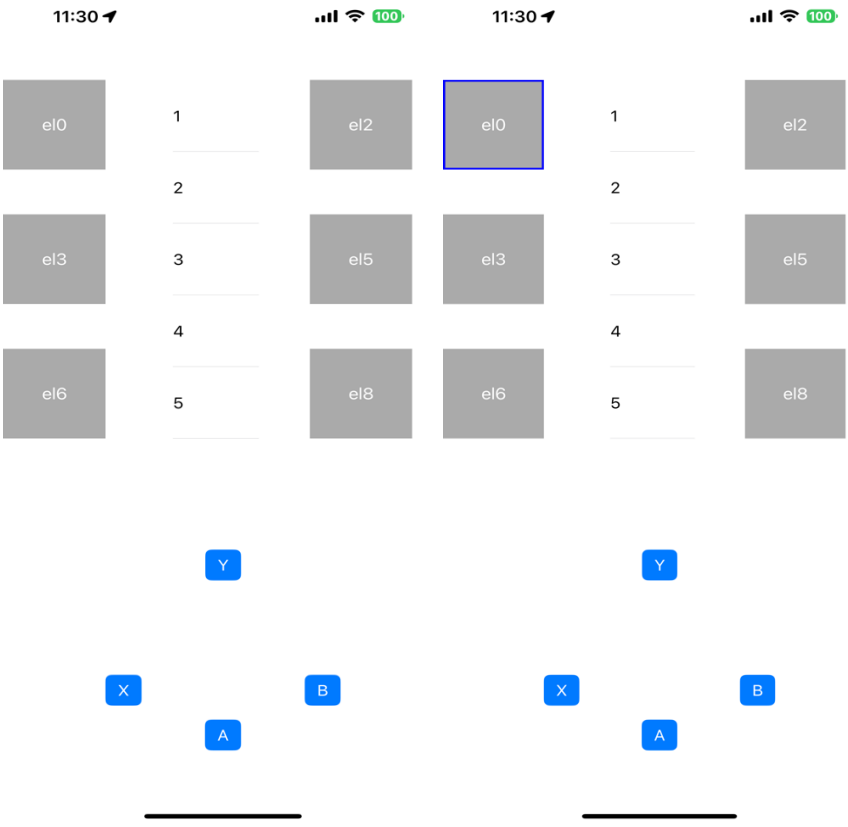

Рис. 3.3, 3.4 Зовнішній вигляд додатку без та з підключенним геймпадом

```swift
override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    do {
        let executionTime = try measureElapsedTime {
            controllerBinder.registerAllFocusableSubviews(from: view)
            controllerBinder.setupDefaultDirectionalInputHandlers()
            controllerBinder.setupTapActionInputHandler()
        }
                print(String(format: "Execution time: %.3f ms", executionTime))
            } catch {
                print("An error occurred: \(error)")
            }

}
func measureElapsedTime(_ operation: () throws -> Void) throws -> Double {
    let startTime = DispatchTime.now()
    try operation()
    let endTime = DispatchTime.now()
    let elapsedTime = endTime.uptimeNanoseconds - startTime.uptimeNanoseconds
    let elapsedTimeInMilliSeconds = Double(elapsedTime) / 1_000_000.0
    return elapsedTimeInMilliSeconds
}
```

```
Execution time: 2.032 ms
```

Рис. 3.5, 3.6 Метод який вираховує кількість витраченого часу для виконання методів та час їх виконання

Отже, якщо ігровий контролер не підключений, жоден з елементів не у фокусі, проте як тільки ігровий контролер підключений, то оскільки як видно з рис 3.5 в коді не було прописано, який саме елемент повинен бути у фокусі, найвищий елемент серед усіх, в даному випадку це кнопка з надписом «el0» стала сфокусованою, оскільки рамка елементу перестала бути прозорою, а забарвилась у синій колір, це чітко видно по рис 3.3 та 3.4. Сам же процес побудови навігаційної мапи та налаштувань кнопок геймпаду для керування додатком зайняв трохи більше 2 мілісекунд, що є доволі позитивним результатом, оскільки це демонструє наскільки швидко працює алгоритм будування мапи після якого можна використовувати геймпад для керування інтерфейсом додатку.

Для того, щоб пересвідчитись у правильності роботи алгоритму будування навігаційної мапи екрану6 після роботи алгоритму, отже тоді коли всі елементи посортовані горизонтально, для кожного елементу було виведено найближчі елементи

для кожного з напрямків, а для того, щоб було легче це перевірити з реальним розташуванням елементів, як на рис 3.3, попередньо для кодного з цих елементів було надано назву, таку саму який надпис є у цього елементу в самому додатку, якщо це кнопки, а єдина UITableView була так і названа TableView.

```
For el0 : Nearest top - Edge, nearestBottom - el3, nearestLeft - Edge, nearestRight - TableView
For el3 : Nearest top - el0, nearestBottom - el6, nearestLeft - Edge, nearestRight - TableView
For el6 : Nearest top - el3, nearestBottom - YButton, nearestLeft - Edge, nearestRight - TableView
For XButton : Nearest top - YButton, nearestBottom - AButton, nearestLeft - Edge, nearestRight - BButton
For TableView : Nearest top - Edge, nearestBottom - YButton, nearestLeft - el0, nearestRight - el2
For AButton : Nearest top - XButton, nearestBottom - Edge, nearestLeft - Edge, nearestRight - Edge
For YButton : Nearest top - TableView, nearestBottom - XButton, nearestLeft - Edge, nearestRight - Edge
For BButton : Nearest top - YButton, nearestBottom - AButton, nearestLeft - XButton, nearestRight - Edge
For el2 : Nearest top - Edge, nearestBottom - el5, nearestLeft - TableView, nearestRight - Edge
For el5 : Nearest top - el2, nearestBottom - el8, nearestLeft - TableView, nearestRight - Edge
For el8 : Nearest top - el5, nearestBottom - YButton, nearestLeft - TableView, nearestRight - Edge
```

Рис. 3.7 Виведена в консоль інформація про всі найближчі елементи для кожного елементу

3.3 Висновки

Підсумовуючи, варто відзначити необхідність створення повної документації перед публікуванням фреймворку в загальний доступ для того, щоб в подальшому використанні інші розробники мали уявлення про те, які завдання вирішує даний фреймворк, та як використовувати його загальнодоступні методи або класи. Окрім цього, для більш легкого і доступного інтегрування фреймворку, важливо забезпечити доступ до нього з декількох менеджерів залежностей, таких як Swift Package Manager та CocoaPods.

Також, перевірка роботи фреймворку, а саме його навігацийної мапи на невеликому тестовому додатку продемонструвало ефективність та правильність роботи фреймворку, що означатиме що за допомогою його можна використовувати ігровий контролер для керування користувацьким інтефейсом додатку і взаємодіяти з ним.

Висновки

Результатом даної роботи стало створення загальнодоступного фреймворку GameControllerBinder на мові програмування Swift для більш доступної інтеграції ігрових контролерів у додатки iOS, а також для можливості керування інтерфейсом додатку за їх допомогою. В роботі було розглянуто принцип роботи фреймворків, а також наявний на даний момент фреймворк для інтеграції ігрових контролерів від Apple – Game Controller. Окрім цього було продемонстровано процес публікування готового фреймворку на GitHub з подальшою його інтеграцією з доступними менеджерами залежностей в Swift.

Розроблений алгоритм в класі FocusManager швидко та ефективно обчислює найближчих сусідів для кожного з елементів, які можуть бути у фокусі, тим самим будуючи навігаційну мапу екрану, для її подальшого використання для корректного пересування між елементами інтерфейсу. Загальнодоступні методи класу GameControllerBinder дозволяють легко з’єднувати будь-які входи контролера до певних дій в самому додатку, а для більш чіткого розуміння можливих входів контролера було створено п’ять різних загальнодоступних перерахувань з описом кожного з них. Запропонований протокол Focusable дозволяє різним елементам бути керованими за допомогою ігрового контролера, а також зовнішньо видозмінюватись коли ці елементи знаходяться у фокусі.

В ході досліджень було розглянуто можливості для подальшого вдосконалення фреймворку, для того, щоб в подальшому розширити сферу застосування цього фреймворку на додатки розроблені за допомогою SwiftUI. Окрім цього, ішні запропоновані нововведення можуть перетворити розроблений фреймворк на універсальний інструмент для інтеграції ігрових контролерів в додатки iOS.

Список використаних джерел

1. iPhone 15 Pro [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.apple.com/iphone-15-pro/.
2. Selway J. Every Game Available on the iPhone 15 Pro So Far [Електронний ресурс] / Jake Selway. – 2023. – Режим доступу до ресурсу: https://gamerant.com/iphone-15-confirmed-games-resident-evil-division-assassins-creed/.
3. Swift [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/swift/.
4. Shirizada R. Protocol-Oriented Programming in Swift: A Comprehensive Guide [Електронний ресурс] / Rashad Shirizada. – 2023. – Режим доступу до ресурсу: https://medium.com/codex/protocol-oriented-programming-in-swift-a-comprehensive-guide-a19ac103d686.
5. What are Frameworks? [Електронний ресурс]. – 2013. – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPFrameworks/Concepts/WhatAreFrameworks.html.
6. Opolentisima L. A Visual History Of Video Game Controllers From 1962 To Now [Електронний ресурс] / Lyle Opolentisima. – 2022. – Режим доступу до ресурсу: https://www.dailyinfographic.com/a-visual-history-of-video-game-controllers.
7. Документація Game Controller [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/gamecontroller.
8. Hollemans M. Mixins and traits in Swift 2.0 [Електронний ресурс] / Matthijs Hollemans. – 2015. – Режим доступу до ресурсу: https://machinethink.net/blog/mixins-and-traits-in-swift-2.0/.
9. Vorona O. Swift && Furious. Coordinates System and position. [Електронний ресурс] / Olga Vorona. – 2019. – Режим доступу до ресурсу: https://medium.com/@olgavorona/swift-furious-coordinates-system-and-position-ca3b9062b282.

10. Creating a Framework for iOS [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.kodeco.com/17753301-creating-a-framework-for-ios.